Appendix 9. Amount of Support Ordered Pseudocode

Step 1. Convert all payments to monthly values. Because support orders can be paid over a wide range of time periods, it is necessary to select one time measurement so that all orders can be compared accurately. For the purpose of this example, a business year of 360 days is used.

Input Fields Required	Output Fields Created / Modified
NCP_Data.Case-ID NCP_Data.Order-Frequency NCP_Data.Amount_of_Support_Ordered NCP_Data.Billing_Status	Support.Case-ID Temp.Order-Frequency Temp. Amount_of_Support_Ordered Support.TMonthly_Amount Support.Billing_Status Support.Date_Modified
Pseudocode	**Reason**
SELECT NCP_Data.Case_ID, NCP_Data.[Order-Frequency],	Square brackets [ ] around field name act as quotation marks, otherwise the hyphen in the field name would be interpreted as a minus sign.
NCP_Data.Amount_of_Support_Ordered FROM NCP_Data; For Each NCP_Data DO Case Select of NCP_Data[Order-Frequency]	
"A": Support.TMonthly_Amount = INT(NCP_Data.Amount_of_Support_Ordered/12)	A = Annually; divide amount by 12 and return the integer value
"B": Support.TMonthly_Amount = INT((NCP_Data.Amount_of_Support_Ordered/14)*30)	B = Biweekly; divide by 14, then multiply the result by 30 and return the integer value
"E": Support.TMonthly_Amount = INT(NCP_Data.Amount_of_Support_Ordered/6)	E = Semiannually; divide by 6 and return the integer value
"Q": Support.TMonthly_Amount = INT(NCP_Data.Amount_of_Support_Ordered/3)	Q = Quarterly; divide by 3 and return the integer value
"M": Support.TMonthly_Amount = INT(NCP_Data.Amount_of_Support_Ordered)	M = Monthly; return the integer value
"S": Support.TMonthly_Amount = INT(NCP_Data.Amount_of_Support_Ordered*2)	S = Semimonthly; multiply by 2 and return integer value
"W": Support.TMonthly_Amount = INT((NCP_Data.Amount_of_Support_Ordered/7)*30)	W = Weekly; divide biweekly by 7, then multiply the result by 30 and return the integer value
OTHER : Writeln(ErrorLog, "Order Frequency out of range. Case ID: ",NCP_Data.Case-ID, " Frequency: ",NCP_Data.[Order-Frequency])	If Frequency is outside range, make error log entry.
Next Record END Case	Break out and return to top of loop.

Support.Case-ID = NCP_Data.Case-ID
Temp.Order-Frequency = NCP_Data.[Order-Frequency]
DNCP_Data.Order-Freq-Amount =
NCP_Data.Amount_of_Support_Ordered
Support.Billing_Status = NCP_Data.Billing_Status
Support.Date_Modified = NOW()
	Update Support Record.

Next Record Done	Loop until all records have been acted on.

Example

Input

Case_ID	Order-Frequency	Order-Freq-Amount	Billing_Status
45447	Quarterly	$150.83	Current
45456	Weekly	$42.50	Delinquent
45457	Biweekly	$196.00	Enforcement

Output

Case_ID	Date_Modified	Order-Frequency	Amount	Monthly	Status
45446	6/11/2002 1:26:56 PM	Monthly	$129.50	$130.00	Current
45447	6/11/2002 1:26:56 PM	Quarterly	$150.83	$50.00	Current
45456	6/11/2002 1:26:56 PM	Weekly	$42.50	$182.00	Delinquent
45457	6/11/2002 1:26:56 PM	Biweekly	$196.00	$420.00	Enforcement

Step 2. Determine range of payment amounts and number of NCPs for each amount.

Input Fields Required	Output Fields Created / Modified
Support.TMonthly_Amount	Report
Pseudocode	**Reason**

SELECT
Support.TMonthly_Amount, Count(Support.TMonthly_Amount) AS CountOfMonthly_Amount
 	
FROM Support;
 	
GROUP BY Support.TMonthly_Amount;
	Same as for NCP Age Count

Example

Input

Monthly_Amount
$50.00
$182.00
$420.00
$130.00

Output

Monthly_Amount	CountOfMonthly_Amount
$50.00	1
$130.00	1
$182.00	1
$420.00	1

After this report is generated, a determination must be made on how to distribute the count of amounts across the entire NCP population. As with the NCP_Age distributions, low values of monthly amounts with a corresponding low number of NCPs paying the amounts might be considered outriders. The same is true for the high end of the distribution. The goal is to distribute the amounts in a manner that will yield meaningful, actionable discriminators.

NOTE: The activities performed before this point are necessary to develop the design of the data mart. The steps that follow normally occur in the ETL process. They are included here to maintain the flow of the discussion.

Step 3. Code Support.TAmount_Code based on distribution plan developed.

Input Fields Required	Output Fields Created/Modified
Support.Case-ID Support.TMonthly_Amount Support.Date_Modified	Support.TAmount_Code Support.Date_Modified
Pseudocode	**Reason**

SELECT
Support.Case-ID,
Support.TMonthly_Amount, Support.Date_Modified,
FROM Support;	

For Each Support, DO Case Select Support.TMonthly_Amount of 1..100: Support.TAmount_Code = 100 101..200: Support.TAmount_Code = 200	
201..300: Support.TAmount_Code = 300 301..400: Support.TAmount_Code = 400 401..500: Support.TAmount_Code = 500 501..600: Support.TAmount_Code = 600	The dividing points and the code were arbitrarily chosen. The code 100 could just as easily represent values from 51 to 151. Numeric codes were chosen because they reflect the content of the data more accurately.
601..700: Support.TAmount_Code = 700 701..800: Support.TAmount_Code = 800 801..900: Support.TAmount_Code = 900 901..1000: Support.TAmount_Code = 1000 Other: Writeln(ErrorLog, "Monthly Amount not within specified ranges. Case ID: ", Support.Case-ID, " Amount: "Support.TMonthly_Amount)	
Next Record	Break out and return to top of loop.
End Case Support.Date_Modified = Now()	
Next Record	Continue to loop back to the program until all records have been processed.

Example

Input

Case_ID	Date_Modified	Monthly_Amount
45447	6/11/2002 1:26:56 PM	$50.00
45446	6/11/2002 1:26:56 PM	$130.00
45456	6/11/2002 1:26:56 PM	$182.00
45457	6/11/2002 1:26:56 PM	$420.00

Output

Case_ID	Monthly_Amount	Amount_Code	Date_Modified
45447	$50.00	100	6/11/2002 3:33:21 PM
45446	$130.00	200	6/11/2002 3:33:21 PM
45456	$182.00	200	6/11/2002 3:33:21 PM
45457	$420.00	500	6/11/2002 3:33:21 PM

Step 4. Distribute Order Amount based on billing status and amount code. This process is fully within the data mart proper and represents the first pieces of information retrieved from it. There are at least three ways to design this step. Each has its advantages and disadvantages.

Methods for Developing Order Amount Table		
Method	**Advantages**	**Disadvantages**
Dynamic—The table is generated and connected to the dimension tables when query is executed.	Information presented is most up-to-date information available.	High system load. Actual value depends on the number of records that have to be acted on. More complex to develop. Higher-order tools required.
Aggregation Record	Faster response. Less complex. Lower-order tools can be used.	Data reloaded on a scheduled basis and so is not necessarily the most current. Moderate system load, but usually performed in off-peak hours. Less flexibility. Ranges embedded with field names.
Example: Code_100_total_Count, Code_100_Paying, Code_100_Non-paying,… Code_N_total_Count, Code_N_Paying, Code_N_Non-paying		
Summary Records	Faster response. Simple structure. Highly flexible. Lowest-order tools can be used	Data reloaded on a scheduled basis and so is not necessarily the most current. Moderate system load, but usually performed in off-peak hours. Repeated read-writes can be eliminated through the use of arrays for a minor increase in complexity.
Example: Code_ID (value stored within the field), Total_Count, Paying_Count, Non_Paying_Count. 1 record for each code amount.		

The final portion of pseudocode for this process will be developed using the Summary Records Model.

Step 5. Load Summary Records

Input Fields Required	Output Fields Created/Modified
Support.Case-ID	DNCP_Amount_Ordered_Summary.Code
Support.TAmount_Code	DNCP_Amount_Ordered_Summary.Total_Count
Support.Billing_Status	DNCP_Amount_Ordered_Summary.Paying_Count
	DNCP_Amount_Ordered_Summary.Non_Paying_Count
	DNCP_Amount_Ordered_Summary.Date_Created
	DNCP_Amount_Ordered_Summary.Date_Modified

Pseudocode (Comments Follow the "//")

```
Delete existing DNCP_Amount_Ordered_Summary Records // Ensure no false counts happen. Start with clean slate.
Establish Array[1..N] of Summary_Record Type // Summary record type is a mirror image of DNCP_Amount_Ordered_Summary. An array is created in order to reduce the number of read / writes to the hard disk. After all of the Support records are read the array will be written out to DNCP_Amount_Ordered_Summary records. N represents the total number of Amount Codes previously created. In our example this will be 10.
ArrayTop = N*100 // This is a check value that prevents the program from trying to access a non-existent array value. There are many static ways and dynamic functions available in various products to achieve the same result. This is the most rudimentary method and is included here to serve as a reminder to always protect the array boundaries.
ArrayBottom = 1 //Same purpose as ArrayTop, protects lower boundary of array.
Initialize all values in Summary_Record Array to 0 // Set all counters in array to known value.
For I = 1 to N
Summary_Record[I].Code = I * 100 // Initialize the values in the Code field for each iteration of the array. Note the use of mathematical functions to establish array element values. Because the amount codes in this example range from 100 to 1,000 in increments of 100, it requires simple math operations to place values into the array and calculate the correct array index. More complex amount codes will require additional manipulation to implement this indexing method.
Next I
Select Support
For Each Support DO

IF Support.TAmount_Code > ArrayTop then
Writeln(ErrorLog, "Amount Code higher than expected. Case ID: ", DNCP_Case_ID, " Amount_Code: ", Support_Amount_Code) //Amount code exceeds ArrayTop value
Next Record //Break out and return to top of loop

IF Support.TAmount_Code < ArrayBottom then
Writeln(ErrorLog, "Amount Code lower than expected. Case ID: ", DNCP_Case_ID, " Amount_Code: ", Support_Amount_Code) // Amount code less than ArrayBottom value
Next Record // Break out and return to top of loop

Summary_Record[Support.TAmount_Code/100].count = Summary_Record[Support.TAmount_Code/100].count + 1 // Using the value in Amount_Code to select the index in the Summary_Record Array. For example an Amount_Code of 100 would evaluate to 1.
IF Support.Billing_Status = "Current" then
 Summary_Record[Support.TAmount_Code/100].Paying_count =
 Summary_Record[Support.TAmount_Code/100].Paying_count + 1
Else// If Billing_Status is equal to "Current" then increment the paying count, else increment the nonpaying count.
 Summary_Record[Support.TAmount_Code/100].Non_Paying_count =
 Summary_Record[Support.TAmount_Code/100].Non_Paying_count + 1
Next Record // Return to the top of the loop and process the next record. Repeat until all Support records have be processed.

For I = 1 to N \\ Write the results out to the Amount_Ordered_Summary Table
DNCP_Amount_Ordered_Summary.Code = Summary_Record[I].Code
DNCP_Amount_Ordered_Summary.Total_count = Summary_Record[I].count
DNCP_Amount_Ordered_Summary.Paying_Count = Summary_Record[I].Paying_count
DNCP_Amount_Ordered_Summary.Non_Paying_Count = Summary_Record[I].Non_Paying_count
DNCP_Amount_Ordered_Summary.Date_Created = Now()
DNCP_Amount_Ordered_Summary.Date_Modified = Now()
NEXT N
DONE
```

The Results

Amount of Support Ordered										
Non_paying	66	112	144	146	150	121	250	200	90	60
Paying	101	212	313	356	195	112	150	100	60	40
Total	167	324	457	502	345	233	400	300	150	100
Amount	$100	$200	$300	$400	$500	$600	$700	$800	$900	$1,000